Today's Plan

1) Interval Scheduling Problem

2) Intermission
   ← Announcements
   ← Reductions

3) Greedy Paradigm
   ← Greedy Stays Ahead
   ← Solution to Interval Scheduling
Classic Motivation:
* single central processor
* many job requests

Question: How do we schedule the jobs?
Classic Motivation:
* single central processor
* many job requests

Question: How do we schedule the jobs?

Details
- Each job has a proposed start time & finish time
- Processor can handle at most 1 job at a time
- Assumption: jobs have equal priority
Interval Scheduling Problem.

Given: List of n jobs, specified by [start, finish] time

\[
\left\langle [s_1, f_1], [s_2, f_2], \ldots, [s_n, f_n] \right\rangle
\]

Goal: Return a set of non-conflicting jobs of maximum cardinality.
Interval Scheduling Problem.

Given: List of \( n \) jobs, specified by \([\text{start}, \text{finish}]\) time intervals:
\[
\left\langle [s_1, f_1], [s_2, f_2], \ldots, [s_n, f_n] \right\rangle
\]

Goal: Return a set of non-conflicting jobs of maximum cardinality with equal priority.
Interval Scheduling Problem.

Given: List of $n$ jobs, specified by $[\text{start}, \text{finish}]$ time

\[
\langle [s_1, f_1], [s_2, f_2], \ldots, [s_n, f_n] \rangle
\]

Goal: Return a set of non-conflicting jobs of maximum cardinality

Defn. A set of intervals $S$ is non-conflicting if for all $i \neq j \in S$

\[S_i \leq S_j \implies f_i < S_j\]
Interval Scheduling Problem

Given: List of \( n \) jobs, specified by \([\text{start}, \text{finish}]\) time
\[
\langle [s_1, f_1], [s_2, f_2], \ldots, [s_n, f_n] \rangle
\]

Goal: Return a set of non-conflicting jobs of maximum cardinality

Defn. A set of intervals \( S \) is non-conflicting if for all \( i \neq j \in S \)
\[ S_i \leq S_j \implies f_i < S_j \]

\( S_i \quad \text{and} \quad f_i \quad S_j \quad \text{and} \quad f_j \) non-conflicting
Interval Scheduling Problem

**Given.** List of n jobs, specified by \([\text{start, finish}]\) time

\[
\langle [s_1, f_1], [s_2, f_2], \ldots, [s_n, f_n] \rangle
\]

**Goal.** Return a set of non-conflicting jobs of maximum cardinality

**Defn.** A set of intervals \(S\) is **non-conflicting** if for all \(i \neq j \in S\)

\[ s_i \leq s_j \implies f_i < s_j \]

---

\(s_i\) \(f_i\) non-conflicting

\(s_k\) \(f_k\) conflicting
Announcements

* HW 1
  - Available on Canvas
  - Submission on Gradescope: Open

* Significant Collaborators
  - Groups of ≤3 total
  - See Ed #38 for partner search #26

* Enrollment in 4820 is capped
  - From CIS: No enrollment increase
A Note on Reductions

Problem $P$ reduces to Problem $Q$ if given an algorithm $A_Q$ that solves $Q$, there exists an algorithm $A_P$ (which makes calls to $A_Q$) that solves $P$.

4820 Philosophy: Look for Reductions!

* Useful for solving HW

* SPOILER: Reductions play essential role in showing certain problems are HARD.
Interval Scheduling Problem.

Given: List of n jobs, specified by [start, finish] time

\[ ([s_1, f_1], [s_2, f_2], \ldots, [s_n, f_n]) \]

Goal: Return a set of non-conflicting jobs of maximum cardinality

Example Applications?

i.e. What other problems reduce to Interval Scheduling?
Greedy Algorithms

Design Paradigm: Choose solution "greedily"
Greedy Algorithms

Design Paradigm. Choose solution “greedily”

Myopic → local decisions that “look good”
Irrevocable → once we make a decision, we never reconsider.
Greedy Algorithms

Design Paradigm: Choose solution "greedily"

- Myopic → local decisions that "look good"
- Irrevocable → once we make a decision, we never reconsider.

Advantage: Fast & Local
Greedy Algorithms

Design Paradigm. Choose solution “greedily”

Myopic → local decisions that “look good”
Irrevocable → once we make a decision, we never reconsider.

Advantage. Fast & Local

Greedy Prototype

1. Sort by “priority”

2. Iterate through elems in priority order
   → Make decision about elem.
Greedy Algorithms

Design Paradigm. Choose solution “greedily”

Myopic → local decisions that “look good”
Irrevocable → once we make a decision, we never reconsider.

Advantage. Fast & Local

Greedy Prototype

1) Sort by “priority”
   \( O(n \log n) \)

2) Iterate through elems in priority order
   Make decision about elem. \( O(1) \)

\[ n \times O(1) = O(n) \]

RT Dominated by sorting!
Greedy Algorithms

Design Paradigm. Choose solution "greedily"

Myopic → local decisions that "look good"
Irrevocable → once we make a decision, we never reconsider.

Warning. Challenging to analyze correctness
Greedy Algorithms

Design Paradigm. Choose solution "greedily"

Myopic $\rightarrow$ local decisions that "look good"
Irrevocable $\rightarrow$ once we make a decision, we never reconsider.

Warning. Challenging to analyze correctness

Often, Greedy approaches are incorrect.
Interval Scheduling Problem

Given list of \( n \) jobs, specified by \([\text{start}, \text{finish}]\) time

\( \langle [s_1, f_1], [s_2, f_2], \ldots, [s_n, f_n] \rangle \)

Candidate "Priority" Functions.
(1) Earliest Start Time

(2) Shortest Interval

(3) # of conflicts

ONLY 2 conflicts
Earliest Finish Time

1. Sort jobs by finish time

2. Schedule = ∅

   Iterate through jobs in sorted order j=1...n
   - if job j does not conflict w/ Schedule
     \[ \text{Schedule} \leftarrow \text{Schedule} \cup \{j\} \]

Return Schedule.
Earliest Finish Time

1. Sort jobs by finish time
2. Schedule = ∅

Iterate through jobs in sorted order j=1...n
- if job j does not conflict with Schedule
  → Schedule ← Schedule ∪ t_j

Return Schedule.
Earliest Finish Time

1. Sort jobs by finish time
2. Schedule = \emptyset

Iterate through jobs in sorted order j = 1, ..., n
- if job j does not conflict with Schedule
  \[ \text{Schedule} \leftarrow \text{Schedule} \cup \{j\} \]

Return Schedule.

Claim. EFT can be implemented with RT \( O(n \log n) \).
Theorem. EFT returns a maximum cardinality set of non-conflicting jobs.

Non-conflicting $\rightarrow$ By design
Theorem. EFT returns a maximum cardinality set of non-conflicting jobs.

Maximum Cardinality $\rightarrow$ "Greedy stays ahead"
Greedy Stays ahead

- Imagine some optimal schedule $O^*$
- Compare output of EFT to $O^*$
Greedy Stays ahead

- Imagine some optimal schedule $O^*$
- Compare output of EFT to $O^*$

$O^*$: $s_1^* f_1^* s_2^* f_2^* h \cdots h s_k^* f_k^*$

EFT: $s_1 f_1 s_2 f_2 \cdots s_2 f_2$
Greedy Stays ahead

- Imagine some optimal schedule $O^*$
- Compare output of EFT to $O^*$

$O^*$:

$S_1^* \quad f_1^* \quad S_2^* \quad f_2^* \quad \cdots \quad S_n^* \quad f_n^*$

EFT:

$s_1 \quad f_1 \quad s_2 \quad f_2 \quad \cdots \quad s_e \quad f_e$

Convenient Notation. Assume jobs are sorted by finishing time.

$f_1^* < f_2^* < \cdots < f_n^*$

$f_1 < f_2 < \cdots < f_e$
Argue that:

(a) if EFT "stays ahead" of $O^*$, then EFT is also optimal

(b) EFT "stays ahead"
Argue that:
(a) if EFT "stays ahead" of $O^*$, then EFT is also optimal
(b) EFT "stays ahead"

WARNING: Need to define "stays ahead" per problem
Claim. EFT "stays ahead" of any $O^*$ in the finish time of the $j$th job.
Optimal schedule \( O^* = \langle [s^*_1, t^*_1], [s^*_2, t^*_2], \ldots, [s^*_k, t^*_k] \rangle \)

EFT schedule \( S = \langle [s_1, t_1], [s_2, t_2], \ldots, [s_k, t_k] \rangle \)

Greedy Stays Ahead Lemma. (part (b))

Let \( S \) be the schedule returned by EFT.

For any optimal \( O^* \),

For all \( j \in \{1, \ldots, |S|\} \)

\( f_j \leq f^*_j \)

"EFT stays ahead of \( O^* \)"
Greedy Stays Ahead Lemma. (part 1(b))

Let \( \mathcal{S} \) be the schedule returned by EFT. For any optimal \( \mathcal{O}^* \),

For all \( j \in \{1, \ldots, |\mathcal{S}|\} \)

\[ f_j \leq f_j^* \]

Pf. By induction on intervals added by EFT.
Greedy Stays Ahead Lemma. (part 1(b))

Let $S$ be the schedule returned by EFT. For any optimal $S^*$, for all $j \in I_1, \ldots, I_{1\gamma}$, $f_j \leq f_j^*$. 

Pf. By induction on intervals added by EFT.

Induction step. (hypothesis holds for all $k' < k$)

Why is this not possible?
Optimal schedule \(O^* = \langle [s_1^*, f_1^*], [s_2^*, f_2^*], \ldots, [s_k^*, f_k^*] \rangle\)

EFT schedule \(S = \langle [s_1, f_1], [s_2, f_2], \ldots, [s_k, f_k] \rangle\)

**Earliest Finish Time Lemma**

[\textit{part (a)}]

If for all \(j \in \{1, \ldots, k\}\), \(f_j \leq f_j^*\)

Then \(S\) is also optimal.

"If EFT stays ahead, then EFT is optimal"
Earliest Finish Time Lemma  \([\text{part (a)}]\)

If for all \(j \in \{1, \ldots, k\}\) \(f_j \leq f_j^*\)\(,\)

Then \(S\) is also optimal.

PT: By contradiction.
Earliest Finish Time Lemma  

If for all $j \in \{1, \ldots, k\}$ \( f_j \leq f_j^* \)

Then $S$ is also optimal.

Proof: By contradiction.